Review #7

Language Support for Fast and Reliable Message-based Communication in Singularity OS

M. Fähndrich, M. Aiken, C. Hawblitzel, O. Hodson, G. Hunt, J. R. Larus, S. Levi

Proceedings of the EuroSys 2006 Conference, 2006

Jean-Pierre Lozi

May 6, 2010

1 Problem

Most mainstream OSes use shared memory as their main means of communication. Message-based communication is an attractive alternative, since it potentially allows for stronger specification and cleaner separation between components. However, it also has major drawbacks, in particular: 1) worse performance since data is usually copied instead of shared and 2) more complicated programming due to the event-driven paradigm in which the flow of the program is not sequential. It would be useful to overcome these drawbacks to allow message-based communication to be used more widely.

2 Solution

Singularity is an operating system structured as a micro-kernel that uses message passing as its sole means of communication between processes. It overcomes the drawbacks of message-based communication mentioned earlier by using a new programming language named Sing# that provides efficient messaging capabilities coupled with verification techniques that make it possible to enforce strong system-wide invariants to facilitate debugging. Sing# also makes it easier to develop user applications than traditional low-level languages like C or C++: it is a type-safe, garbage-collected and feature-rich language based on C#.

Messages are exchanged over bi-directional channels. These channels can also be used to exchange channel endpoints or pointers to memory location. This last point is crucial: it allows for fast data transfers by removing the burden of copying memory blocks. Furthermore, the OS enforces *channel contracts*. A contract describes the messages, message argument types and valid message interaction sequences for a channel as a finite state machine. Contracts allow for a cleaner, better specified isolation between components and therefore facilitate debugging.

Since processes are isolated and individually garbage-collected, passing data from one thread to another could be an obstacle to safe garbage collection: when a message is sent, references to message arguments or transfered data blocks can be found in both the sending and receiving threads, for instance. To solve this issue, the Sing# compiler statically checks that processes only access memory that they own and that a memory block always belongs to a single thread at any given time. To this end, data on the GC heap is separated from data in the *exchange heap*—the heap that is used to exchange all data between processes (messages and memory pointed to by exchanged pointers). At any time, each memory block is owned by a function in a process: ownership is automatically passed from a function (and thread) to another through simple rules—when a variable is passed as a parameter, its ownership is passed to the called function for the whole time of its execution, for example. The only exception to this automatic ownership management strategy is for memory blocks from the exchange heap whose pointers are

exchanged *via* messages: a special type of object, *TCell*, is provided to track them explicitly. This whole data-passing mechanism allows for easier application development by asserting that there are no memory leaks.

The use of Sing# offers another advantage: its strong type-checking capabilities allow the whole system to dismiss hardware memory protection altogether.

3 Evaluation

Performance experiments show that Singularity is faster than Linux and Windows for certain tasks. Performing a system call, switching between two threads, and, more importantly, sending a message are faster on Singularity. Moreover, the authors show that sending a block of memory from one thread to another is fast and does not depend on the size of the block. Linux and Windows are slower at performing this task, especially for larger blocks, since they copy the data instead of just passing a pointer.

4 Benefits

Singularity integrates an advanced garbage-collected language and statically verified contracts, allowing for safe message passing between processes. Moreover, it features safe zero-copy passing of memory blocks, thereby allowing developers to transmit data from one process to another at low cost. Also, the use of software memory protection allows for better performance. All of these features are easy to use thanks to special constructs and features provided by the language.

5 Shortcomings

All user programs have to be rewritten in Sing# to be run on Singularity (for now, at least). Moreover, Singularity might suffer from the usual problems microkernels have, in particular regarding performance: the benchmarks are not really fair since they use message-passing for Linux and Windows even though these OSes generally use faster shared-memory based IPCs.